Maximum Knowledge Orthogonality Reconstruction with Gradients in Federated Learning

Feng Wang, Senem Velipasalar, and M. Cenk Gursoy
EECS Department, Syracuse University, Syracuse, NY, 13244
{fwang26, svelipas, mcgursoy}@syr.edu*

Abstract

Federated learning (FL) aims at keeping client data local to preserve privacy. Instead of gathering the data itself, the server only collects aggregated gradient updates from clients. Following the popularity of FL, there has been considerable amount of work revealing the vulnerability of FL approaches by reconstructing the input data from gradient updates. Yet, most existing works assume an FL setting with unrealistically small batch size, and have poor image quality when the batch size is large. Other works modify the neural network architectures or parameters to the point of being suspicious, and thus, can be detected by clients. Moreover, most of them can only reconstruct one sample input from a large batch. To address these limitations, we propose a novel and analytical approach, referred to as the maximum knowledge orthogonality reconstruction (MKOR), to reconstruct clients' data. Our proposed method reconstructs a mathematically proven high-quality image from large batches. MKOR only requires the server to send secretly modified parameters to clients and can efficiently and inconspicuously reconstruct images from clients' gradient updates. We evaluate MKOR's performance on MNIST, CIFAR-100, and ImageNet datasets and compare it with the state-of-the-art baselines. The results show that MKOR outperforms the existing approaches, and draw attention to a pressing need for further research on the privacy protection of FL so that comprehensive defense approaches can be developed. The code is available at: https://github.com/wfwf10/MKOR.

1. Introduction

Federated learning (FL), as a type of distributed learning, has attracted growing interest from both academia and industry. In the FL framework, multiple clients transmit gradient updates, instead of their data, to preserve client's privacy. In a typical setting, a parameter server broadcasts current network parameters to a subset of individual clients. Each selected client generates the gradient update based on its own local data batch and uploads it back to the server. Then, the server aggregates the gradient updates from selected clients and iterates until convergence [4, 13, 14].

However, it has been shown that FL is vulnerable to adversarial privacy attacks [9, 33]. By receiving gradient updates, an adversarial server can gain information about or reconstruct the private input data. Some threat models consider an "honest-but-curious" server that trains the model normally while recovering input data from unrealistically small batches [9, 33]. Other approaches consider a malicious server that modifies the network structure or parameters to deal with larger batches [7, 29], but these suspicious modifications might be detected and rejected by the clients.

Motivated by the aforementioned limitations of existing works, we propose a novel and analytical approach, referred to as the maximum knowledge orthogonality reconstruction (MKOR), to reconstruct better quality input data with larger batches. During common FL training, clients upload batches of gradients to a parameter server, to cooperatively train a neural network [13]. We consider a malicious server that uses the original structure of a convolutional neural network (CNN), but modifies the parameters instead to maximize the orthogonality between prior- and post-knowledge. The general assumption is that neighboring pixels have similar intensity values, which we define as the prior-knowledge. By receiving gradient updates for broadcasted network parameters, a malicious server obtains further information about the unknown data, which we define as the post-knowledge. The server optimizes reconstruction performance for multiple inputs from large batches. We propose a malicious parameter design on the fully-connected (FC) classifier to decouple a small subset of nodes, which broadcasts analytic attacks from a single FC layer to multiple FC layers. Then, we define the knowledge orthogonality to judge the efficiency of malicious parameters in convolutional layers, and demonstrate the corresponding parameter design on the convolutional layers in a VGG model [23] to optimize reconstruction. We also demonstrate how these modifications can be performed inconspicuously to avoid being detected by clients, and show that such modifications can easily apply to other CNNs with similar architectures, such as LeNet [18]. We evaluate the performance of MKOR on MNIST dataset [6] with LeNet5 network, and CIFAR-100 [15] and ImageNet [5] datasets with the VGG16 network, then compare it with existing works.

*This work was supported in part by the National Science Foundation (NSF) under Grant 2221875 and New York State Department of Economic Development under Contract Number C080116.

Main contributions of this work include the following:

- In contrast to most optimization-based attacks, MKOR can recover private data from arbitrarily large batch sizes.
- Unlike optimization- and semi-optimization-based attacks, MKOR is completely analytical, and provides guaranteed high-fidelity reconstruction from arbitrary data distribution. We calculate the mathematical solution without iterative optimization or generating gradient updates.
- In contrast to malicious threat models, which suspiciously modifies a network's structure, MKOR uses the original architecture, which is VGG16 or LeNet in this paper.
- Compared to malicious threat models, which only modify the parameters, (i) MKOR is inconspicuous and hard to detect, since there is not a large number of zero weights dominating any layer, and the sum of gradients is not dominated by any sample (there is no weight changing within one batch for different samples either); and (ii) MKOR is more efficient, since it generates multiple reconstructions with one gradient update.

2. Related Work

We consider a common FL framework, where a parameter server orchestrates a set of U clients to cooperatively train a neural network [13]. Each client u has a local dataset with K_u images, where the k^{th} image $\mathbf{x}_{u,k}$ is assigned the label $y_{u,k}$. The goal of FL is to iteratively optimize the neural network parameters $\mathbf{A}$ (the weights and biases) to minimize the sum of loss $\mathcal{L}$ via $\operatorname{argmin}_{\mathbf{A}} \sum_u^U \sum_k^{K_u} \mathcal{L}(\mathbf{x}_{u,k}, y_{u,k} \,|\, \mathbf{A})$, where each client u uploads the local sum of gradients $\sum_k^{K_u} \nabla_{\mathbf{A}} \mathcal{L}(\mathbf{x}_{u,k}, y_{u,k} \,|\, \mathbf{A})$, instead of original data $\cup_k^{K_u} \{\mathbf{x}_{u,k}, y_{u,k}\}$, to the server. In the remainder of this paper, we consider the data $\{\mathbf{x}_k, y_k\}$ and the gradients of each single client batch with size K, and omit the client index u. The list of notations is provided in Tab. 1 in the Suppl. file.

There have been different works [9,20,26,33] suspecting the central server to be curious or malicious with the goal of obtaining clients' data. While some methods [20, 26] investigate reconstructing the labels y_k from gradients, many studies demonstrate a privacy attack that reconstructs the client input $\mathbf{x}_k$ from the gradients, which is usually referred to as the "gradient inversion" [9] or "deep leakage" [33]. Most of these works consider the threat of an "honest-but-curious" server, which not only maintains the FL functionality and updates the parameters as planned, but also intends to reconstruct client inputs from the received gradients. After an optimization-based attack was proposed in [28], majority of the ensuing attacks were designed in the same fashion [9, 30, 31, 33]. Given the well-trained parameters $\mathbf{A}$, these attacks randomly initialize dummy data $\mathbf{x}'_k$ and labels y'_k, generate dummy gradient $\sum_k^K \nabla_{\mathbf{A}} \mathcal{L}(\mathbf{x}'_k, y'_k \,|\, \mathbf{A})$, and iteratively update the dummy data with its dummy gradient.

Yet, the aforementioned approaches can only reconstruct from gradients of batches with very limited batch size. This restricts their effectiveness and usability in realistic settings, where the gradient is aggregated over many samples [13]. There are other assumptions undermining their practicality. For instance, they are less likely to succeed in the absence of FL convergence, and some of them make a strong assumption about knowing the Batch-Norm statistics and private labels [12]. It should also be noted that optimization-based methods have no guarantee of convergence. While the best performing sample over a dataset can be recognizable, the performance on other samples can be disappointing. Moreover, some other works [11, 30], including generative adversarial network (GAN)-based attacks [19], use a pre-trained network to reconstruct high-resolution images in relatively small batches where the batch sizes are far less than the number of classes. Most of these methods have strong assumption on the correlation between the distributions of the attacker's training dataset and the user dataset, which is barely true in FL. In contrast, with MKOR, the attacker does not need surrogate data.

Other threat models consider a malicious server that can manipulate the network structure or parameters. Yet, these modifications can be deemed suspicious and easily detectable. Privacy attacks in [3, 7] construct large FC layers whose output size is similar to the input image, and reconstruct inputs with a large batch size. Yet, such a large FC layer, which is larger than a typical CNN, is highly suspicious and thus can be rejected, since the purpose of using conv. layers is to reduce the FC layer size. Attacks in [21] and the feature fishing attack in [29] even require the server to send different malicious parameters to the same client, to be applied on different samples in the same batch. Such modification is too suspicious and easily detectable. The class fishing attack in [29] keeps the network structure intact, and modifies the weights in the last FC layer by setting all but those for the target label to zero. Hence, the sum gradient is dominated by one sample and can be easily detected. For example on ImageNet, it sets 99.9% of weights to zero and 99.9% of images have zero gradient, while our MKOR only modifies at most 14% of weights to non-zero values in a layer and every image has similar (non-zero) gradient magnitude. The attack in [17] requires the index of each sample as input (which is not necessary for FL). It also requires excessively large number of iterations on the same user with the same training batch, while most clients might only appear once in cross-device FL of that batch size [13].

It should be noted that, against privacy attacks, every client should verify the received network before deploying their data. Thus, when an anomaly is detected, clients may exit the FL process before uploading the gradients. The detection methods include checking the size of the FC layer, checking if the majority of weights are 0 in each layer, and sampling the gradients to see if the sum gradient is dominated by a few of them. These detection processes are easy

Table 1. Comparison between different methods of input reconstruction attack

Methods	Type	Assumption on server	Maximum batch size	Samples recovered per batch	Generalization to unknown datasets
Gradient inversion [9]	Optimization	Honest-but-curious	≤ 8	≤ 8	Trained on client data
Deep leakage [33]	Optimization	Honest-but-curious	≤ 8	≤ 8	Trained on client data
Other earlier works [28, 30, 31]	Optimization	Honest-but-curious	≤ 8	≤ 8	Restricted
Modify network structure [3, 7]	Analytical	Malicious-very easy to detect	> 100	varies	Data-free (no training)
Class fishing [29]	Optimization	Malicious-easy to detect	> 100	≤ 1	Restricted
MKOR (Ours)	Analytical	Malicious-and-inconspicuous	> 100	≤ 100	Data-free (no training)

to implement, and can detect all the aforementioned attacks. We also note that data distribution is mostly hidden from the server, and most of the optimization-based methods depend on the well-trained parameters with the clients' data.

In this work, we also focus on the reconstruction of the input $\mathbf{x}_k$ from the gradients, and demonstrate that MKOR can inconspicuously achieve guaranteed reconstruction performance on large batches with original/unmodified network models without any assumptions on data distribution. A general comparison of our approach with the aforementioned methods is provided in Tab. 1, illustrating the advantages of MKOR addressing the prior limitations.

3. Maximum Knowledge Orthogonality Reconstruction

We introduce MKOR as a novel and fully analytical method to reconstruct inputs $\mathbf{x}_k$ from the sum gradient $\sum_k^K \mathbf{G}_k$, where $\mathbf{G}_k = \nabla_{\mathbf{A}} \mathcal{L}(\mathbf{x}_k, y_k \,|\, \mathbf{A})$. We consider a malicious server that modifies the parameters $\mathbf{A}$ of a CNN and can handle large batches. The proposed MKOR is inconspicuous and avoids being detected by clients. We explain details by using the original VGG16 architecture [23], and then apply it on other networks such as LeNet.

3.1. Sum Gradient Knowledge Shrinkage

We first show the necessity to decouple gradients. Given a certain distribution of the gradient vector $\mathbf{G}$, we can denote the information leakage about $\mathbf{G}_1$, caused by revealing the sum gradient, as $H(\mathbf{G}_1) - H(\mathbf{G}_1|\sum_{k=1}^K \mathbf{G}_k)$, where $H(\cdot)$ is the entropy. Compared to a scenario with a small batch size, where leakage is significant, it is obvious that as batch size $K \to \infty$, $\lim_{K\to\infty} \frac{1}{K}\sum_{k=1}^K \mathbf{G}_k = \lim_{K\to\infty} \frac{1}{K-1}\sum_{k=2}^K \mathbf{G}_k = \mathbb{E}(\mathbf{G})$, where $\mathbb{E}(\cdot)$ denotes the expectation. Hence,

$$\lim_{K\to\infty} \left(H(\mathbf{G}_1) - H(\mathbf{G}_1|\sum_{k=1}^K \mathbf{G}_k) \right) = 0, \tag{1}$$

and the knowledge leakage from batch-normalized average gradient goes to 0 (except the parameters in the last FC layer and the decoupled parameter n in l^{th} layer, where $\exists k, \forall k' \neq k, \mathbf{G}^l_{k'}[n] = 0$. Decoupling all the parameters can be detected by clients easily and is not recommended. This will be further explained in Sec. 3.2).

Specifically, if we assume that each element $\mathbf{g} \in \mathbf{G}$ follows a Gaussian distribution $\mathcal{N}(\mu_g, \sigma_g^2)$, then each element in sum gradient $\sum_{k=1}^K \mathbf{g}_k \sim \mathcal{N}(K\mu_g, K\sigma_g^2)$, and $H(\sum_{k=1}^K \mathbf{g}_k) = \frac{1}{2}\log(2\pi K\sigma_g^2) + \frac{1}{2}$. Then, for $K > 1$:

$$H(\mathbf{g}_1) - H(\mathbf{g}_1|\sum_{k=1}^K \mathbf{g}_k) = H(\sum_{k=1}^K \mathbf{g}_k) - H(\sum_{k=1}^K \mathbf{g}_k|\mathbf{g}_1) = O\left(\frac{1}{K}\right) \tag{2}$$

where $O(\frac{1}{K})$ denotes K^{-1} complexity. The derivation is provided in the Suppl. material.

Thus, when the batch size is large, it is necessary to decouple the parameters so that a subset of the gradients originates from samples with the same label. We next introduce our design modifying the FC classifier parameters to achieve the best reconstruction performance.

3.2. FC Classifier Input Reconstruction

First, we note that a linear combination of an FC classifier input from each class can be reconstructed. Consider a "vanilla" feed-forward FC classifier, which can be regarded as a first-order Markov process. By reconstructing the input of the FC classifier, we will have all the information we need to reconstruct the input of the convolutional part.

Also, there can be ample amount of samples such that they are the only ones representing their class in the batch, and thus, the aforementioned reconstruction becomes not just a linear combination, but can uniquely recover the sample information. Assume that the local sample distribution over classes of a client is $\mathbf{p} = \{p_1, p_2, \ldots, p_N\}$, where N is the number of labels. The expectation of the number of samples that are the only ones representing their class is

$$\begin{aligned} &\sum_{n=1}^N \sum_{k=1}^K \left(\mathrm{P}(y_k = n) \prod_{k'\neq k}^K \mathrm{P}(y_{k'} \neq n) \right) \\ &= \sum_{n=1}^N \sum_{k=1}^K p_n(1-p_n)^{K-1} = K\sum_{n=1}^N p_n(1-p_n)^{K-1}. \end{aligned} \tag{3}$$

For example, if there are K=100 samples for N=100 classes with a uniform distribution, we expect 36.97 unique samples. Thus, we can reconstruct a significant amount of samples that are not combined with others. We now explain how the input is reconstructed without error. Let L denote the number of layers and each layer l has N^l input nodes (most superscripts indicate the layer index as listed in Tab. 1 in the Suppl. file). The weights and biases are denoted by $\mathbf{W}^l \in \mathbb{R}^{N^{l+1}\times N^l}$ and $\mathbf{b}^l \in \mathbb{R}^{N^{l+1}}$, where the parameters $\mathbf{A}^l = \{\mathbf{W}^l, \mathbf{b}^l\}$ and $N_{L+1} \overset{\text{def}}{=} N$. For each layer l, we denote the input of the FC classifier from sample k as $\mathbf{z}^l_k \in \mathbb{R}^{N^l}$, and thus the output of conv. layers is $\mathbf{z}^1_k \in \mathbb{R}^{N_1}$.

3.2.1 Single Layer

If the FC classifier consists of only one layer without an activation function, it is easy to recover the input without

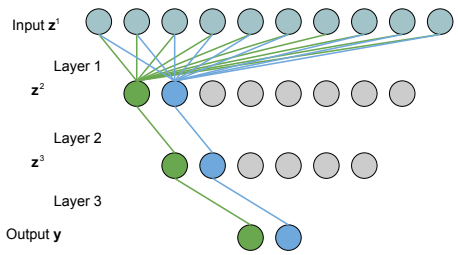

Figure 1. A naive approach to decouple the second layer of multi-layer FC classifier.

modifying the parameters [9, 32]. For the k^{th} sample with output vector $\mathbf{y}_k$ with assigned label y_k,

$$\frac{\partial \mathcal{L}(\mathbf{x}_k, y_k \,|\, \mathbf{A})}{\partial \mathbf{W}^1} = \frac{\partial \mathcal{L}(\mathbf{x}_k, y_k \,|\, \mathbf{A})}{\partial \mathbf{y}_k} \frac{\partial \mathbf{y}_k}{\partial \mathbf{W}^1} = \frac{\partial \mathcal{L}(\mathbf{x}_k, y_k \,|\, \mathbf{A})}{\partial \mathbf{b}^1} \mathbf{z}_k^{1\mathrm{T}}, \quad (4)$$

where $\frac{\partial \mathcal{L}(\mathbf{x}_k, y_k \,|\, \mathbf{A})}{\partial \mathbf{y}_k} = \frac{\partial \mathcal{L}(\mathbf{x}_k, y_k \,|\, \mathbf{A})}{\partial \mathbf{b}^1}$ since $\mathbf{y}_k = \mathbf{W}^1 \mathbf{z}_k^1 + \mathbf{b}^1$. Hence, we have

$$\mathbf{z}_k^1 = \frac{\partial \mathcal{L}(\mathbf{x}_k, y_k \,|\, \mathbf{A})}{\partial \mathbf{W}^1[y_k, :]} \Big/ \frac{\partial \mathcal{L}(\mathbf{x}_k, y_k \,|\, \mathbf{A})}{\partial \mathbf{b}^1[y_k]}, \quad (5)$$

where $\mathbf{W}^1[y_k, :]$ denotes the y_k^{th} row of $\mathbf{W}^1$, and the division is element-wise. In practice, an attacker may obtain $\sum_{k=1}^{K} \frac{\partial \mathcal{L}(\mathbf{x}_k, y_k \,|\, \mathbf{A})}{\partial \mathbf{A}}$ instead of that for a single sample, so the attacker will reconstruct

$$\hat{\mathbf{z}}_n^1 = \sum_{k=1}^{K} \frac{\partial \mathcal{L}(\mathbf{x}_k, y_k \,|\, \mathbf{A})}{\partial \mathbf{W}^1[y_k, :]} \Big/ \sum_{k=1}^{K} \frac{\partial \mathcal{L}(\mathbf{x}_k, y_k \,|\, \mathbf{A})}{\partial \mathbf{b}^1[y_k]}, \quad (6)$$

where $\hat{\mathbf{z}}_n^1 = \mathbf{z}_k^1 \neq \mathbf{0}$ if and only if there is exactly one k such that $y_k = n$. Thus, the input of a linear layer can be accurately reconstructed. The reason this reconstruction method works is that there exists one **decoupled** element in the output vector $\mathbf{y}_k$ for each label n, i.e.,

$$\sum_{\substack{k=1 \\ y_k = n}}^{K} \frac{\partial \mathcal{L}(\mathbf{x}_k, y_k \,|\, \mathbf{A})}{\partial \mathbf{y}_k[n]} \neq \mathbf{0}, \quad \sum_{\substack{k=1 \\ y_k \neq n}}^{K} \frac{\partial \mathcal{L}(\mathbf{x}_k, y_k \,|\, \mathbf{A})}{\partial \mathbf{y}_k[n]} = \mathbf{0}. \quad (7)$$

3.2.2 Multiple Layers - Naive Decoupling

While gradients in a single-layer classifier are easily decoupled, parameter modification is required to decouple the output of the first layer for a multi-layer classifier. A naive and also suspicious approach to decouple the output $\mathbf{z}_k^2$ of the first layer in an FC classifier with multiple layers is illustrated in Fig. 1. If all weights, except the connected ones, are set to zero and the back-propagation is not zeroed out by the activation function, we have the following for each label $n \in \{1, 2, \ldots, N\}$:

$$\sum_{\substack{k=1 \\ y_k = n}}^{K} \frac{\partial \mathcal{L}(\mathbf{x}_k, y_k \,|\, \mathbf{A})}{\partial \mathbf{z}_k^2[n]} \neq \mathbf{0}, \quad \sum_{\substack{k=1 \\ y_k \neq n}}^{K} \frac{\partial \mathcal{L}(\mathbf{x}_k, y_k \,|\, \mathbf{A})}{\partial \mathbf{z}_k^2[n]} = \mathbf{0}. \quad (8)$$

Similar to Eq. (4)(5)(6), we have

$$\hat{\mathbf{z}}_n^1 = \sum_{k=1}^{K} \frac{\partial \mathcal{L}(\mathbf{x}_k, y_k \,|\, \mathbf{A})}{\partial \mathbf{W}^1[y_k, :]} \Big/ \sum_{k=1}^{K} \frac{\partial \mathcal{L}(\mathbf{x}_k, y_k \,|\, \mathbf{A})}{\partial \mathbf{b}^1[y_k]}. \quad (9)$$

While this naive approach decouples N elements in $\mathbf{z}_k^2$, and thus, can reconstruct the input $\hat{\mathbf{z}}_n^1$, the modification on the parameters with the given pattern and massive number of zeros are too suspicious and can easily be detected by clients. Hence, we propose an inconspicuous decoupling approach next, which achieves the same result.

3.2.3 Multiple Layers - Inconspicuous Decoupling

Consider a multi-layer FC classifier with ReLU activation. Our goal is to decouple a set of nodes $\{n_1, n_2, \ldots, n_N\}$ in $\mathbf{z}_k^2$, where $\mathbf{z}_k^2[n_i]$ uniquely corresponds to one output label as described in (8). While all indices should be picked randomly, we list them in order (i.e., $n_1 \leq 2 < n_2 \leq 4 < \ldots < n_N \leq 2N$, similar to Fig. 1) for easier understanding. The parameters in the first layer are modified to

$$\mathbf{W}^1[n, :] = \begin{cases} \alpha_n \mathbf{W}^1[n-1, :], \text{ if } n \leq 2N, n \bmod 2 = 0 \\ \mathbf{W}^1[n, :], \text{ otherwise} \end{cases} \quad (10)$$

$$\mathbf{b}^1[n] = \begin{cases} \alpha_n \mathbf{b}^1[n-1], \text{ if } n \leq 2N, n \bmod 2 = 0 \\ \mathbf{b}^1[n], \text{ otherwise} \end{cases} \quad (11)$$

where $\alpha_n < 0$ is a constant, N is the number of labels, and $n \bmod 2 = 0$ denotes n to be an even number. Thus, it is guaranteed to have N different indices $n \leq 2N$, where each of them indexes a non-zero node $\mathbf{z}_k^2[n]$ given any unknown sample k. For each following layer $l > 1$, the parameters are modified to

$$\mathbf{W}^l[n^{l+1}, n^l] = \begin{cases} \left|\mathbf{W}^l[n^{l+1}, n^l]\right|, \text{ if } l = 2, \lfloor \frac{n^l}{2} \rfloor = n^{l+1} \leq N \\ \mathcal{N}(0,\, \sigma^2), \text{ if } l = 2, \lfloor \frac{n^l}{2} \rfloor \neq n^{l+1}, n^l \leq 2N \\ \left|\mathbf{W}^l[n^{l+1}, n^l]\right|, \text{ if } l > 2, n^l = n^{l+1} \leq N \\ \mathcal{N}(0,\, \sigma^2), \text{ if } l > 2, n^l \neq n^{l+1}, n^l \leq N \\ \mathbf{W}^l[n^{l+1}, n^l], \text{ otherwise} \end{cases} \quad (12)$$

$$\mathbf{b}^1[n^{l+1}] = \begin{cases} \left|\mathbf{b}^1[n^{l+1}]\right|, \text{ if } n^{l+1} \leq N \\ \mathbf{b}^1[n^{l+1}], \text{ otherwise} \end{cases} \quad (13)$$

where $|\cdot|$ denotes the absolute value, $\lfloor \cdot \rfloor$ denotes rounding down, and $\mathcal{N}(0,\, \sigma^2)$ denotes independent and identically distributed (i.i.d.) Gaussian noise with zero mean and variance $\sigma^2 \ll 1$. Eq. (12) ensures that first $2N$ nodes in the 2nd layer and first N nodes in the following layers are decoupled to avoid any desired node being zeroed out by ReLU activation. Thus, each output element $\mathbf{y}_k[n]$ has a unique back-propagation path to $\mathbf{z}_k^2[2n-1]$ and $\mathbf{z}_k^2[2n]$, and one of them is guaranteed to be decoupled for any sample. Since the majority of the parameters in (12) and (13) are not modified and the statistics of all parameters, such as mean and variance, are almost unchanged, it is extremely hard to detect the modification via parameters. Also, we note that such decoupling only affects a small subset of parameters and is hard to detect via gradient values, since the ReLU activation naturally eliminates gradients from many nodes. For better defense against any potential pattern-based detection, we may add a small i.i.d. noise to every parameter.

With the reconstructed input features $\hat{\mathbf{z}}_n^1$, it is possible to train an auto-encoder, use the encoder as the convolutional part for a CNN, and use the decoder to reconstruct the input image $\hat{\mathbf{x}}_n$. Yet, the performance highly depends on the

similarity of the local training set and the client dataset distributions. Instead, we introduce the analytical malicious parameter approach without any assumptions on data distribution. Before explaining how we set the malicious parameters of the convolutional layers and reconstruct the image, we first introduce the concept of knowledge orthogonality.

3.3. Knowledge Orthogonality

If gradients from none of the samples are zeroed out (otherwise the server will be suspicious), the gradient update of conv layers cannot be decoupled, and will be a summation over all samples. In Eq. (1)(2), we proved that the knowledge from such sum of gradients decreases and converges to zero as the batch size increases, which is the reason that most existing approaches do not perform well with large batches. In our approach, we alternatively use the extracted features $\hat{\mathbf{z}}_n^1$ from conv layers with malicious parameters to reconstruct the input $\hat{\mathbf{x}}_n$ for each label n, and introduce the knowledge orthogonality to judge the malicious parameters.

While there are edges and noise, images are mostly smooth, i.e., it is highly probable that neighboring pixels have similar intensity values [16, 22, 25]. Also, the distribution of the intensities x and x' of two neighboring pixels from a set of images is likely to have the highest deviation in the direction $(1,1)$. This eigenvector indicates that $x+x'$ has higher variance, while the variance of $x-x'$, with weights $(1,-1)$ in the orthogonal direction, is much lower. Thus, we know more about an image by knowing "$x+x'=d_1$" than knowing "$x-x'=d_2$", since the latter is already known to some extent by the smoothness assumption. We consider the fact "neighboring pixels tend to be alike" as our prior-knowledge, and the constant value $d_1 = ax + a'x'$ as our post-knowledge, which is obtained from our reconstructed $\hat{\mathbf{z}}_n^1$. When the vector (a, a') is close to the first eigenvector $(1,1)$ and orthogonal to $(1,-1)$ that corresponds to the prior-knowledge, we obtain the most information.

In practice, for most existing CNNs, given the fact that $|\mathbf{z}^1| < |\mathbf{x}|$, where $|\cdot|$ denotes the length of vector, we will not be able to get a unique solution for all elements of input $\mathbf{x}$. Instead, we want to set malicious parameters such that each element in $\mathbf{z}^1$ reveals the sum of a group of neighboring pixels, and thus, we achieve the knowledge orthogonality and reconstruct a blurred image.

3.4. Convolutional Blocks Input Reconstruction

While there are many constraints to get the sum of neighboring pixels in the popular models, we first explain how we modify parameters to approximate the local sum. Specifically, we show our design on the VGG16 [23] structure.

3.4.1 The Naive Approach

In Fig. 2, we sketch a naive approach to assign the malicious parameters on the VGG convolutional layers in a smaller scale for easier understanding. In this naive approach, the connected input and output channels have the convolutional filter parameters as assigned in the diagram, and all other parameters are set to zero. For the 3 RGB input channels, layer ① divides the information of each channel into two different channels. The "copy" filter copies the information, and the "min" filter generates the reversed value (we assume each input element is a floating number with maximum value 1). Later on, in the input reconstruction part, we will use the former to get the local maximum, and the latter to get the local minimum. Layer ② and layer ③ copy all the considered input channels, while layer ③ also has max-pooling with stride 2. Layer ④ divides each considered input channel into four channels. Each of the four filters copies the same information, or shifts the input features by one pixel to the right, down, and lower right. If there has been i max-pooling layers before this layer, such filter will shift the considered region in the input by 2^i pixels. Finally, layer ⑤ is the max-pooling layer and flattens the output to a one-dimensional vector $\mathbf{z}^1$ as the input of the FC classifier. We denote the tensor before flatten as $\mathbf{z}^0$, which has the same set of elements but different shape than $\mathbf{z}^1$.

3.4.2 Input Reconstruction

We now explain how to reconstruct input $\hat{\mathbf{x}}_n$ from output $\hat{\mathbf{z}}_n^1$ that is reconstructed from the FC classifier. For each sample, each element in $\mathbf{z}^0$ denotes a local maximum or a local minimum within a certain region in input $\mathbf{x}$. Assuming there are I-many 4-direction layers and J-many max-pooling layers with only "copy" layer, the region size is $2^{I+J} \times 2^{I+J}$, and is much smaller than the receptive field. For VGG, the output $\mathbf{z}^0$ has shape $7\times7\times512$, so each element $\mathbf{z}^0[h,w,c]$ in the 6 channels, corresponding to only "copy" filters, indicates the local maximum or minimum of a color in the considered region of the input $\mathbf{x}$:

$$\begin{aligned}\mathbf{R}[h,w,c] = \mathbf{x}[&(h-1)2^{I+J}+1 : h2^{I+J},\\ &(w-1)2^{I+J}+1 : w2^{I+J}, \lceil \frac{c}{2\times 4^I} \rceil],\end{aligned} \tag{14}$$

where the index $d_1 : d_2$ denotes "from d_1 to d_2", and $\lceil\cdot\rceil$ denotes rounding up. For each of the other elements $\mathbf{z}^0[h,w,c]$ in the other channels, and for each i^{th} 4-directional layer, the region is shifted to the right by 2^{i+J} pixels if c corresponds to a "right" filter or a "lower right" filter, and we denote as indicator $\mathbf{1}_{\mathrm{r}}(c,i) = 1$, otherwise 0. Also, this region is shifted down by 2^{i+J} pixels if c corresponds to a "lower" filter or a "lower right" filter, and we denote as indicator $\mathbf{1}_{\mathrm{l}}(c,i) = 1$, otherwise 0. Thus, the total shifted distance in input pixels is

$$s_{\mathrm{r}}(c) = \sum_i^I \mathbf{1}_{\mathrm{r}}(c,i)2^{i+J}, \quad s_{\mathrm{l}}(c) = \sum_i^I \mathbf{1}_{\mathrm{l}}(c,i)2^{i+J}, \tag{15}$$

and the shifted considered region can be expressed as

$$\begin{aligned}\mathbf{R}[h,w,c] = \mathbf{x}\Big[&(h-1)2^{I+J}+1+s_{\mathrm{l}}(c) : h2^{I+J}+s_{\mathrm{l}}(c),\\ &(w-1)2^{I+J}+1+s_{\mathrm{r}}(c) : w2^{I+J}+s_{\mathrm{r}}(c), \lceil \frac{c}{2\times 4^I} \rceil\Big].\end{aligned} \tag{16}$$

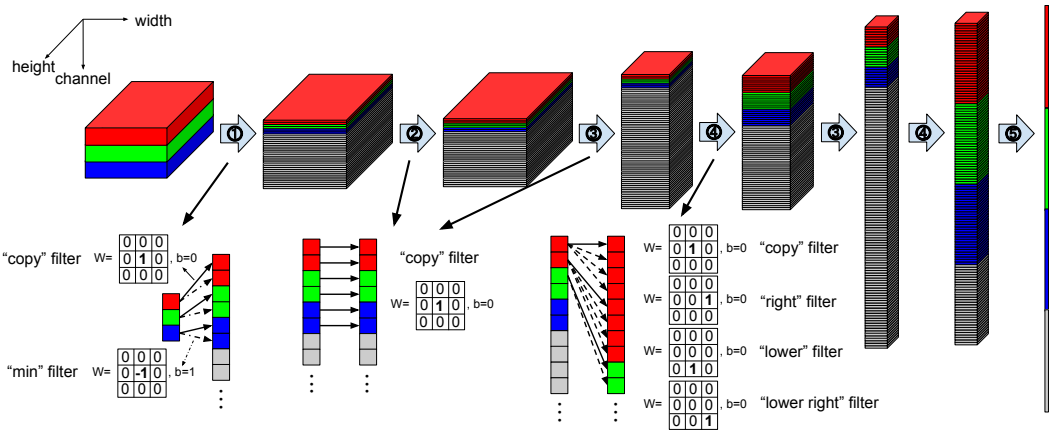

Figure 2. Diagram of the naive approach of setting convolutional layer parameters to maximize knowledge orthogonality and obtain local sum of input $\mathbf{x}$. The colored cuboids denote the input and output channels of each convolutional layer, and only the red, green and blue ones are considered. Layer ① has both "copy" filter and "min" filter, and we only consider the first 6 channels in its output. Layer ② copies the inputs, while layer ③ copies the input and also max-pools. Layer ④ divides each input channel into four output channels and shifts the features to corresponding directions. Layer ⑤ is a max-pooling layer with flattening.

Thus, each input element $\mathbf{x}_n[h', w', c']$ of class n is described by at most 2^{2I} maximum elements and 2^{2I} minimum elements in $\mathbf{z}_n^0$. To get the tightest bound of the unknown element $\mathbf{x}_n[h', w', c']$, we denote its estimated maximum and minimum as

$$\max\left(\mathbf{x}_n[h', w', c']\right) \overset{\text{def}}{=} \min_{\substack{\forall\{c|\mathbf{x}_n[h',w',c']\in\mathbf{R}[h,w,c],\\ \lceil c/4^I\rceil \bmod 2=1\}}} \left(\mathbf{z}_n^0[h, w, c]\right), \tag{17}$$

$$\min\left(\mathbf{x}_n[h', w', c']\right) \overset{\text{def}}{=} \max_{\substack{\forall\{c|\mathbf{x}_n[h',w',c']\in\mathbf{R}[h,w,c],\\ \lceil c/4^I\rceil \bmod 2=0\}}} \left(\mathbf{z}_n^0[h, w, c]\right), \tag{18}$$

where mod denotes the remainder. One way to estimate $\mathbf{x}_n[h', w', c']$ is to take the average of (17) and (18) as our reconstructed input $\hat{\mathbf{x}}_n$:

$$\hat{\mathbf{x}}_n[h', w', c'] \overset{\text{def}}{=} \frac{\max(\mathbf{x}_n[h', w', c']) + \min(\mathbf{x}_n[h', w', c'])}{2}. \tag{19}$$

Any reconstruction method using this malicious parameter setting should have at least 7×7 and at most $7 \cdot 2^I \times 7 \cdot 2^I$ resolution. When the output $\hat{\mathbf{z}}_n^1$ denotes a combination of samples from one class n, the reconstructed input $\hat{\mathbf{x}}_n$ is also a combination of samples $\mathbf{x}_k$. To improve the efficiency of utilizing 512 output channels of $\mathbf{z}^0$, we pick $I = 3$ so that 75% of the channels are considered.

3.4.3 Inconspicuous Approach

We now propose our inconspicuous approach to modify the network parameters so that it achieves the same reconstruction result as above while being more difficult to detect. For each layer, we denote the *set of* considered input and output channels as $\mathcal{C}_{\text{in}}$ and $\mathcal{C}_{\text{out}}$, respectively. The indices of all channels are randomly shuffled, different from Fig. 2. If any of the filters in Fig. 2 is assigned to the network weight $\mathbf{W}[c_{\text{out}}, c_{\text{in}}, :, :]$ and bias $\mathbf{b}[c_{\text{out}}]$ where $c_{\text{in}} \in \mathcal{C}_{\text{in}}$ and $c_{\text{out}} \in \mathcal{C}_{\text{out}}$, we denote this by the indicator function $\mathbf{1}(c_{\text{in}}, c_{\text{out}}) = 1$, and denote the corresponding parameters by $A_{\text{filter}} = \{W, b\}$. Otherwise, $\mathbf{1}(c_{\text{in}}, c_{\text{out}}) = 0$. We use A_{noise} to denote i.i.d. noise parameters with Gaussian distribution with zero mean and variance $\sigma^2 \ll 1$. Therefore, we may assign each set of parameters in the network as

$$A[c_{\text{in}}, c_{\text{out}}] = \begin{cases} \beta A_{\text{filter}}, \text{ if } \mathbf{1}(c_{\text{in}}, c_{\text{out}}) = 1 \\ A_{\text{noise}}, \text{ else if } c_{\text{in}} \in \mathcal{C}_{\text{in}}, c_{\text{out}} \in \mathcal{C}_{\text{out}} \\ A[c_{\text{in}}, c_{\text{out}}], \text{ otherwise} \end{cases} \tag{20}$$

where $\beta > 0$ is a constant, which can be cancelled out during reconstruction. Since both $\mathcal{C}_{\text{in}}$ and $\mathcal{C}_{\text{out}}$ are only a small subset of all channels in most of the layers, the majority of the parameters are not modified, making it hard to detect. To better avoid being detected by any potential pattern-based detection, we can add i.i.d. noise to all modified parameters, or set smaller I (i.e., set less 4-direction layers, and reconstruct with lower resolution). The overall process of MKOR on a VGG network is presented in Algorithm 1.

Algorithm 1 Maximum Knowledge Orthogonality Reconstruction

Server sets the malicious parameters of FC layers according to (10)(11)(12)(13), and sets those of conv layers according to (20)
Server broadcasts modified neural network to each client u
Clients execute:
for client u **do**
  Generate and upload the sum gradient $\sum_k^K \mathbf{G}_k$ to server
Server executes:
for each client u **do**
  for each label n **do**
    Solve $\hat{\mathbf{z}}_n^1$ according to (9)
    $\hat{\mathbf{z}}_n^0 = \text{reshape}(\hat{\mathbf{z}}_n^1)$
    Estimate the input $\hat{\mathbf{x}}_n$ according to (19)

3.5. Other Layers

We now discuss the other types of layers that do not affect the overall function. For the dropout layer in FC classifier, we can set multiple parallel paths to each output label with the same functionality to increase the probability of

one of the paths not being dropped. As for the initial batch normalization and the batch-norm layers in convolutional layers, they only affect the magnitude of reconstruction, i.e., we will get a weighted input estimation $\gamma\hat{\mathbf{x}}_n$, where $\gamma > 0$ depends on the batch-norm statistics. We can calibrate the magnitude by modifying the histogram [10], which may incur mean square error but does not affect human perception.

3.6. Applicability to Other Networks

We can easily apply MKOR to other CNNs with similar architecture, such as LeNet [18]. Two major differences of LeNet from VGG are the Sigmoid activation function instead of ReLU, and average pooling layers instead of max-pooling. Therefore, the "min" filter is no longer necessary, and we can directly reconstruct the input $\hat{\mathbf{x}}_n$ by inverting the Sigmoid activations in each convolutional layer from the features $\hat{\mathbf{z}}_n^1$ of the first FC layer.

4. Experiments

We evaluate our MKOR on the MNIST dataset [6] with the LeNet5 network [18], and on higher resolution images, from CIFAR-100 [15] and ImageNet [5] datasets, with the VGG16 network [23]. We perform comparison with baselines to illustrate the superior performance of MKOR. We consider the challenging cases of large batches, where the batch size is equal to or larger than the class number, without modifying the original network architecture. We consider the case of a single local update, where MKOR performs best. As stated in [27], while more local updates allow higher system throughput, this incurs slightly higher error at convergence. Since no optimization or iterations are needed in MKOR, there are no additional hyper-parameters. MKOR can decode one batch with 100 CIFAR-100 images on VGG16 by a CPU within 2 min. In comparison, optimization-based methods require GPUs and take 8 hours to recover the same batch.

First, we reconstruct batches of 100 MNIST images with LeNet5, which is frequently used in previous work, where the batch size of 100 is larger than the number of classes (10). We compare MKOR with deep leakage from gradient (DLG) [33], improved DLG (iDLG) [31], gradient inversion (GI) [9], and class fishing (CF) [29], since these methods are more suitable for large batches than other existing methods. GI assumes an "honest-but-curious" server, and CF uses malicious parameters without modifying the network architecture. Yet, CF [29] can still be detected by checking for large number of zeros in the last layer. In comparison, MKOR is not only suitable for large batches but also 'inconspicuous' and hard to detect by clients as detailed in Sec. 3.2.3 and 3.4.3. Moreover, we notice that the majority of optimization-based attack methods, including DLG, iDLG, GI and CF, delete some or all of the pooling layers for better attack performance. We also use LeNet as was done in most of the baselines. We perform comparison with both original LeNet and modified LeNet (i.e., use stride=2 in conv layers instead of avg pooling layers). In Tab. 2, a quantitative comparison is presented in terms of the structural similarity index measure (SSIM) and peak signal-to-noise ratio (PSNR) between the original and the reconstructed image. We note that since CF can reconstruct the image of one class only, the average SSIM and average PSNR are not available. Tab. 2 shows the result of 10 random batches. Since the batch size of 100 is greater than the number of classes, images are randomly selected and there are repeated classes. Thus, each cell in Tab. 2 is the average of 1000 images. For a qualitative comparison, we present a reconstructed batch of images by MKOR, GI and CF using original LeNet in Fig. 3. The results of DLG and iDLG, and the images reconstructed with the modified LeNet can be found in the Suppl. material with larger figures. From Tab. 2 and Fig. 3, MKOR outperforms all the baselines, and from Tab. 2 and Fig. 2 in the Suppl. file, MKOR has even better performance when modified LeNet is used, where modification does not involve using a very large FC layer.

Table 2. Comparison of different reconstruction methods on MNIST with LeNet5 for 10 random batches with batch size 100.

max, avg SSIM *max*, avg PSNR	Original LeNet	Modified LeNet
DLG	*0.13*, 0.02, *2.00*, 0.52	*0.08*, 0.01, *1.21*, 0.20
iDLG	*0.11*, 0.03, *1.28*, 0.18	*0.14*, 0.04, *1.29*, 0.26
GI	*0.30*, 0.13, *15.64*, 11.44	*0.41*, 0.18, *15.96*, 11.48
CF	*0.21*, —, *14.63*, —	*0.42*, —, *15.95*, —
MKOR (ours)	**0.54**, **0.27**, **17.06**, **12.79**	**0.69**, **0.38**, **18.72**, **13.53**

Since optimization-based methods rely heavily on the gradient of convolutional layers, they do not perform well on large batches. As explained in Sec. 3.1, reconstruction from large batches should focus more on extracted features.

In Tab. 2 of the Suppl. file, we consider additive Gaussian noise on gradients as used in differential privacy [1]. MKOR still outperforms all the baselines under high noise, which provides higher level of privacy protection. We note that differential privacy and local differential privacy [2] fail under such attack, because these works do not take the neural network structure and its parameters into account. More discussion is provided in the Suppl. file. Since MKOR does not depend on image distribution, transformation-based defenses, such as [8], do not work either, and MKOR can still recover the transformed input. In Tab. 3 of the Suppl. file, we consider Soteria [24], which optimizes the perturbation such that the reconstructed data is severely degraded. As seen, while Soteria is a general defense method not focus-

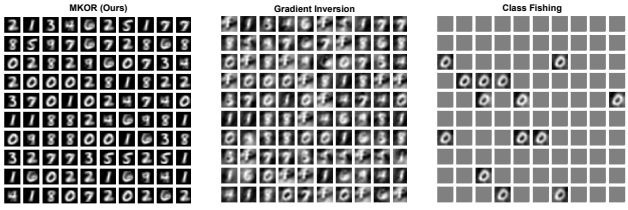

Figure 3. Qualitative comparison of different input reconstruction methods on MNIST with original LeNet5 for a batch size of 100.

Table 3. Comparison of different input reconstruction methods on CIFAR-100. Each data point shows the mean and standard deviation over the same $U = 10$ set of batches.

Methods	Maximum SSIM (↑)	Average SSIM (↑)	Maximum PSNR (↑)	Average PSNR (↑)
GI	0.52 ± 0.04	0.37 ± 0.01	17.51 ± 1.61	11.2 ± 0.36
CF	0.81 ± 0.10	-	21.56 ± 4.65	-
MKOR (Ours)	$\mathbf{0.98 \pm 0.01}$	$\mathbf{0.87 \pm 0.00}$	$\mathbf{33.85 \pm 2.07}$	$\mathbf{24.14 \pm 0.28}$

ing on a specific attack, it is not as effective on MKOR as the other attacks. Please see the Suppl. material for additional details, the images showing the reconstruction under Gaussian noise and Soteria defense, and single large samples for reconstruction detail.

Since GI and CF (although for a single class) performed better in the MNIST experiment with large batch size, we compare MKOR with GI and CF on CIFAR-100 dataset [15] with VGG16 network [23]. We use their experiment setting of unique batches, where each image is a unique sample, being the only one representing its class. The batch size $K=100$, which is the same as the number of classes. The extracted features $\mathbf{z}^0$ from VGG conv. layers are of shape $7\times7\times512$. If an arbitrary function is used (instead of a CNN), we can reconstruct images with 91×91 resolution. Yet, due to the practical limit of the CNN structure, the function is not arbitrary, so our post-knowledge is limited with more information loss. Here, we show the results obtained by Alg. 1. Since we pick the number of 4-direction layers as $I=3$, the expected resolution of MKOR is between 7×7 and $7\cdot2^I\times7\cdot2^I = 56\times56$. In Tab. 3, we show the quantitative comparison. With a batch size of 100, MKOR outperforms both GI and CF. We present a reconstructed batch of images in Fig. 4 and single reconstructed samples for a closer look in Fig. 5 (please refer to Suppl. material for additional and larger images).As can be seen, CF [29] can only reconstruct one image per batch, and the images reconstructed by GI [9] are distorted. In contrast, with MKOR, most images in the batch are better recognizable. Moreover, while modifying the network architecture does not make a difference on GI and CF performance, MKOR with a modified network can accurately reconstruct an arbitrary dataset when there is no image with repeating class. For such unique batch from CIFAR-100, we have [max SSIM, avg SSIM, max PSNR, avg PSNR] = [1.000, 1.000, 163.557, 156.650]. The plots showing the reconstruction of the same batch size K=100, but with less number of classes N can be found in the Suppl. material.

In Tab. 4, we evaluate MKOR with larger batch (1000 in this case) and larger image resolution (224×224 on ImageNet vs. 32×32 with CIFAR100) on VGG16 and modified VGG16 (conv with stride=2 instead of pooling layers) with and without Gaussian noise with normalized standard deviation (10^{-1}). We test MKOR on 10 unique batches and 10 random batches of batch size 1000. While most works do not attempt to use a batch size of 1000 on ImageNet, MKOR is able to maintain excellent performance, and we have not seen similar works on such large batches with unmodified benchmark models. It should be noted that optimization-based methods on this batch size of high-resolution images require prohibitively large memory and long runtime. The images showing the reconstruction with this setting can be found in the Suppl. material.

Table 4. Performance of MKOR with VGG16 on ImageNet with batch size 1000

max, avg SSIM *max*, avg PSNR	Original VGG16 BS=1000	Modified network BS=1000
unique batch, no noise	*0.93*, 0.48, *30.12*, 17.88	*1.00*, 1.00, *inf*, inf
unique batch, Gaussian (10^{-1})	*0.92*, 0.45, *29.99*, 17.50	*0.97*, 0.80, *31.24*, 31.00
random batch, no noise	*0.90*, 0.42, *30.05*, 15.05	*1.00*, 0.72, *inf*, inf
random batch, Gaussian (10^{-1})	*0.88*, 0.41, *26.44*, 14.80	*0.88*, 0.52, *28.37*, 17.63

Figure 4. Qualitative comparison between different input reconstruction methods on CIFAR-100 for a batch size of 100.

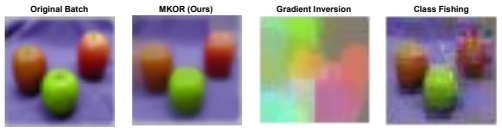

Figure 5. Larger plots of samples from Fig. 4 for reconstruction details.

5. Conclusion

We have introduced the maximum knowledge orthogonality reconstruction (MKOR) as a novel federated learning (FL) input reconstruction method, which can be used with large batches. Our approach maliciously sets the parameters of fully-connected layers to accurately reconstruct the input features, and of convolutional layers to maximize the orthogonality between prior and post knowledge. For both cases, we have proposed an inconspicuous approach to avoid being detected by skeptical clients. Experiments have demonstrated that MKOR outperforms other baselines on large batches of the MNIST dataset with LeNet model, and on CIFAR-100 and ImageNet datasets with a VGG16 model by providing better reconstructed images both qualitatively and quantitatively. Our results encourage further research on the protection of data privacy in FL.

References

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 308–318, 2016. 7

[2] Abhishek Bhowmick, John Duchi, Julien Freudiger, Gaurav Kapoor, and Ryan Rogers. Protection against reconstruction and its applications in private federated learning. *arXiv preprint arXiv:1812.00984*, 2018. 7

[3] Franziska Boenisch, Adam Dziedzic, Roei Schuster, Ali Shahin Shamsabadi, Ilia Shumailov, and Nicolas Papernot. When the curious abandon honesty: Federated learning is not private. *arXiv preprint arXiv:2112.02918*, 2021. 2, 3

[4] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191, 2017. 1

[5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 1, 7

[6] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012. 1, 7

[7] Liam Fowl, Jonas Geiping, Wojtek Czaja, Micah Goldblum, and Tom Goldstein. Robbing the fed: Directly obtaining private data in federated learning with modified models. *arXiv preprint arXiv:2110.13057*, 2021. 1, 2, 3

[8] Wei Gao, Shangwei Guo, Tianwei Zhang, Han Qiu, Yonggang Wen, and Yang Liu. Privacy-preserving collaborative learning with automatic transformation search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 114–123, 2021. 7

[9] Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, and Michael Moeller. Inverting gradients-how easy is it to break privacy in federated learning? *Advances in Neural Information Processing Systems*, 33:16937–16947, 2020. 1, 2, 3, 4, 7, 8

[10] Rafael C Gonzalez. *Digital image processing*. Pearson education india, 2009. 7

[11] Ali Hatamizadeh, Hongxu Yin, Holger R Roth, Wenqi Li, Jan Kautz, Daguang Xu, and Pavlo Molchanov. Gradvit: Gradient inversion of vision transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10021–10030, 2022. 2

[12] Yangsibo Huang, Samyak Gupta, Zhao Song, Kai Li, and Sanjeev Arora. Evaluating gradient inversion attacks and defenses in federated learning. *Advances in Neural Information Processing Systems*, 34:7232–7241, 2021. 2

[13] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1–2):1–210, 2021. 1, 2

[14] Jakub Konečnỳ, H Brendan McMahan, Daniel Ramage, and Peter Richtárik. Federated optimization: Distributed machine learning for on-device intelligence. *arXiv preprint arXiv:1610.02527*, 2016. 1

[15] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 1, 7, 8

[16] David S Lalush and Miles N Wernick. Iterative image reconstruction. In *Emission Tomography*, pages 443–472. Elsevier, 2004. 5

[17] Maximilian Lam, Gu-Yeon Wei, David Brooks, Vijay Janapa Reddi, and Michael Mitzenmacher. Gradient disaggregation: Breaking privacy in federated learning by reconstructing the user participant matrix. In *International Conference on Machine Learning*, pages 5959–5968. PMLR, 2021. 2

[18] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 1, 7

[19] Zhuohang Li, Jiaxin Zhang, Luyang Liu, and Jian Liu. Auditing privacy defenses in federated learning via generative gradient leakage. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10132–10142, 2022. 2

[20] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. Exploiting unintended feature leakage in collaborative learning. In *2019 IEEE symposium on security and privacy (SP)*, pages 691–706. IEEE, 2019. 2

[21] Dario Pasquini, Danilo Francati, and Giuseppe Ateniese. Eluding secure aggregation in federated learning via model inconsistency. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 2429–2443, 2022. 2

[22] Khalid Sayood. *Lossless compression handbook*. Elsevier, 2002. 5

[23] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1, 3, 5, 7, 8

[24] Jingwei Sun, Ang Li, Binghui Wang, Huanrui Yang, Hai Li, and Yiran Chen. Soteria: Provable defense against privacy leakage in federated learning from representation perspective. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9311–9319, 2021. 7

[25] Ch Thum. Measurement of the entropy of an image with application to image focusing. *Optica Acta: International Journal of Optics*, 31(2):203–211, 1984. 5

[26] Aidmar Wainakh, Fabrizio Ventola, Till Müßig, Jens Keim, Carlos Garcia Cordero, Ephraim Zimmer, Tim Grube, Kristian Kersting, and Max Mühlhäuser. User-level label leakage from gradients in federated learning. *Proceedings on Privacy Enhancing Technologies*, 2022(2):227–244, 2022. 2

[27] Jianyu Wang and Gauri Joshi. Cooperative sgd: A unified framework for the design and analysis of local-update sgd algorithms. *The Journal of Mach. Learn. Research*, 22(1):9709–9758, 2021. 7

[28] Zhibo Wang, Mengkai Song, Zhifei Zhang, Yang Song, Qian Wang, and Hairong Qi. Beyond inferring class representatives: User-level privacy leakage from federated learning. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 2512–2520. IEEE, 2019. 2, 3

[29] Yuxin Wen, Jonas Geiping, Liam Fowl, Micah Goldblum, and Tom Goldstein. Fishing for user data in large-batch federated learning via gradient magnification. *arXiv preprint arXiv:2202.00580*, 2022. 1, 2, 3, 7, 8

[30] Hongxu Yin, Arun Mallya, Arash Vahdat, Jose M Alvarez, Jan Kautz, and Pavlo Molchanov. See through gradients: Image batch recovery via gradinversion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16337–16346, 2021. 2, 3

[31] Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. idlg: Improved deep leakage from gradients. *arXiv preprint arXiv:2001.02610*, 2020. 2, 3, 7

[32] Junyi Zhu and Matthew Blaschko. R-gap: Recursive gradient attack on privacy. *arXiv preprint arXiv:2010.07733*, 2020. 4

[33] Ligeng Zhu, Zhijian Liu, and Song Han. Deep leakage from gradients. *Advances in neural information processing systems*, 32, 2019. 1, 2, 3, 7